信息网络与协议

实验指导书

（2025 秋季学期）

课 程 教 师

卢 汉 成　　**hclu@ustc.edu.cn**

助　教

刘 书 艺　　**lsy55@mail.ustc.edu.cn**

魏 思 奇　　**siqiwei@mail.ustc.edu.cn**

注意事项

1) 本课程一共包含 4 个实验，分五周完成。
2) 实验报告通过邮箱进行提交。报告提交截止时间为每周日，迟交会有记录。
3) 第四次实验时间为两周，可以提前完成并提交实验报告。
4) 如在实验过程中遇到问题，请及时通过课程群与助教联系。

实验一：Linux 的网络操作与配置

一、 实验目标

1. 了解 ARP 协议的原理。
2. 了解 IP 协议并学习基础的 IP 协议配置。
3. 了解 TCP/IP 连接过程。

二、 实验原理

1. ARP 协议

ARP 协议是地址解析的通用协议（Address Resolution Protocol），其提供了网络层地址（IP 地址）到物理地址（mac 地址）之间的动态映射，在实际通信中，物理网络使用硬件地址进行报文传输。IP 报文在封装为数据链路层帧进行传送时，就有必要把 IP 地址转换为对应的硬件地址，ARP 正是动态地完成这一功能的。通过 arp -n 或者 ip neigh show 可以查看 ARP 缓存表的内容。

2. IP 协议

IP 协议（Internet Protocol，互联网协议）中的 IP 地址，保证了联网设备的唯一性，实现了网络通信的面向无连接和不可靠的传输功能。 IP 版本有 IPv4 和 IPv6，通过 route -n 和 route --inet6 命令可查看相应的 IP 路由配置。IPv4 中规定 IP 地址长度为 32，最大地址个数为2^{32}；而 IPv6 中 IP 地址的长度为 128，即最大地址个数为2^{128}, 与 IPv4 相比，IPv6 有以下优势:

- IPv6 具有更大的地址空间
- IPv6 使用更小的路由表
- IPv6 增加了增强的组播支持以及对流的控制
- IPv6 加入了对自动配置的支持
- 更好的头部格式

3. TCP/IP

TCP/IP 指传输控制协议/网际协议 (Transmission Control Protocol / Internet Protocol), 其提供面向连接的可靠传输服务，工作在 OSI 的传输层，TCP 工作过程主要是建立连接，然后从应用层程序接收数据并进行传输，采用虚电路的方式进行工作，发送数据前需要在发送接收端建立连接，数据发送后发送方等待接收方做出确认接收的应答，否则发送方任务数据丢失并重新发送该数据。

三、 实验内容

1. 理解 ARP 协议

1） 在 HostA 和 HostB 中分别打开一个终端用于本实验。

2）在 HostA 和 HostB 中分别执行命令 ifconfig eth0，查看并记录它们各自的 IPv4 地址、IPv6 地址 (global 和 link 两种) 以及以太网接口的物理地址。

3）在 HostA 中执行命令 arp -n 或 ip neigh show 查看并记录本机 ARP 缓存表的内容。

4）在 HostA 中执行命令 ping -c 1 HostB 的 IPv4 地址向 HostB 发送 ICMP 请求报文。收到 ICMP 响应后再次执行命令 arp -n 或 ip neigh show 查看 HostA 的 ARP 缓存表的内容。

5）在 HostA 的 ARP 缓存表里面可获得 HostB 的 MAC 地址，记录下来，检查与 HostB 上 ifconfig eth0 命令的执行结果是否一致。

6） 在 HostA 中执行命令 ping -c 1 202.38.64.246，收到 ICMP 响应后继续执行命令 ip neigh show 查看 HostA 的 ARP 缓存表，记录结果。简要解释为何无法看到对应于地址 202.38.64.246 的 ARP 表项而只能得到网关的某网卡的 MAC 地址。提示：思考网段(链路)、广播域的概念。

2. 学习 IP 协议基本配置

（此步骤仅使用虚拟机 HostA）

1） 分别用命令 route -n 和命令 route --inet6 查看本机的 IPv4 和 IPv6 路由配置，记录所在子网的子网掩码/前缀长度，并于前面 ifconfig 的结果作比较。

2）执行以下两个命令分别查看系统内核的 IPv4 和 IPv6 的 FORWARD 值，记录下来。简单解释这个值的含义以及为何这个值是这样设定的。提示：思考主机与路由器的区别。

命令 1：cat /proc/sys/net/ipv4/ip_forward

命令 2：cat /proc/sys/net/ipv6/conf/all/forwarding

3. TCP 端口探测

1）在 HostB 的终端 1 中执行 nc -l 1958 侦听 1958 端口。

2）在 HostA 的终端 1 执行 nc HostB 的 IPv4 地址 1958 。

3） 在 HostA 终端 2 中执行命令 netstat -aunt 来观察自己主机上的所有 TCP 与 UDP 连接状况，将输出的信息记录下来。请在上述记录的结果中找到对应于上述连接的那条记录并解释这条记录的含义。

4）执行命令 nc HostB 的 IPv4 地址 100，记录命令执行结果。同样使用 netstat -aunt 来查看本机的连接状况，请判断这次 telnet 连接是否成功建立并简单说明原因。

四、 思考题

1. 现在有一个网段的 IP 地址和子网掩码分别为 202.38.75.0/255.255.255.192，请计算该网段中一共有多少个全局 IPv4 地址可供主机使用，或者说这个网络中有多少真正可分配的 IP 地址？
2. 实验中执行 ifconfig eth0 查看接口的配置信息时可以观察到一个重要的参数 MTU，请问这个值是多少？查询资料说明 MTU 参数的用途。
3. IPv6 地址长度时 IPv4 地址长度的 4 倍，不过在今后的纯 IPv6 网络环境中路由器的路由表的规模反而有望减小，请简单解释这是为什么？
4. 一条 TCP 连接需要哪几个参数标识？

五、 实验报告

报告要求：PDF 报告，内容包括实验题目、实验原理（对实验中所用到的所有命令加以解释）、实验结果（实验中所有需要记录的结果，截图中需要包含以学号命名的用户名，命令及结果）、思考题、实验收获。

注意 实验报告及时提交到邮箱 ustc_infonet@163.com 中，第一次实验报告提交 PDF 文件，标题和文件命名规则为：姓名-学号-信网第一次实验，晚交会记录并扣分。

实验二：IPv6 基本操作

一、 实验目标

1. 学习使用 tcpdump 抓取 IPv6 路由器公告报文并分析报文信息。
2. 了解 TCP 三次握手的工作过程。
3. 掌握 IPv6 协议的隧道通信。

二、 实验原理

1. TCP 三次握手

所谓三次握手 (Three-way Handshake)，是指建立一个 TCP 连接时，需要客户端和服务器总共发送 3 个包。三次握手的目的是连接服务器指定端口，建立 TCP 连接，并同步连接双方的序列号和确认号，交换 TCP 窗口大小信息。在 socket 编程中，客户端执行 connect() 时，将触发三次握手。

a) 第一次握手 (SYN=1, seq=x): 客户端发送一个 TCP 的 SYN 标志位置 1 的包，指明客户端打算连接的服务器的端口，以及初始序号 X, 保存在包头的序列号 (Sequence Number) 字段里。发送完毕后，客户端进入 SYN_SEND 状态。

b) 第二次握手 (SYN=1, ACK=1, seq=y, ACKnum=x+1): 服务器发回确认包 (ACK) 应答。即 SYN 标志位和 ACK 标志位均为 1。服务器端选择自己 ISN 序列号，放到 Seq 域里，同时将确认序号(Acknowledgement Number) 设置为客户的 ISN 加 1，即 X+1。发送完毕后，服务器端进入 SYN_RCVD 状态。

c) 第三次握手 (ACK=1，ACKnum=y+1): 客户端再次发送确认包 (ACK)，SYN 标志位为 0，ACK 标志位为 1，并且把服务器发来 ACK 的序号字段 +1，放在确定字段中发送给对方，并且在数据段放写 ISN 的 +1。发送完毕后，客户端进入 ESTABLISHED 状态，当服务器端接收到这个包时，也进入 ESTABLISHED 状态，TCP 握手结束。

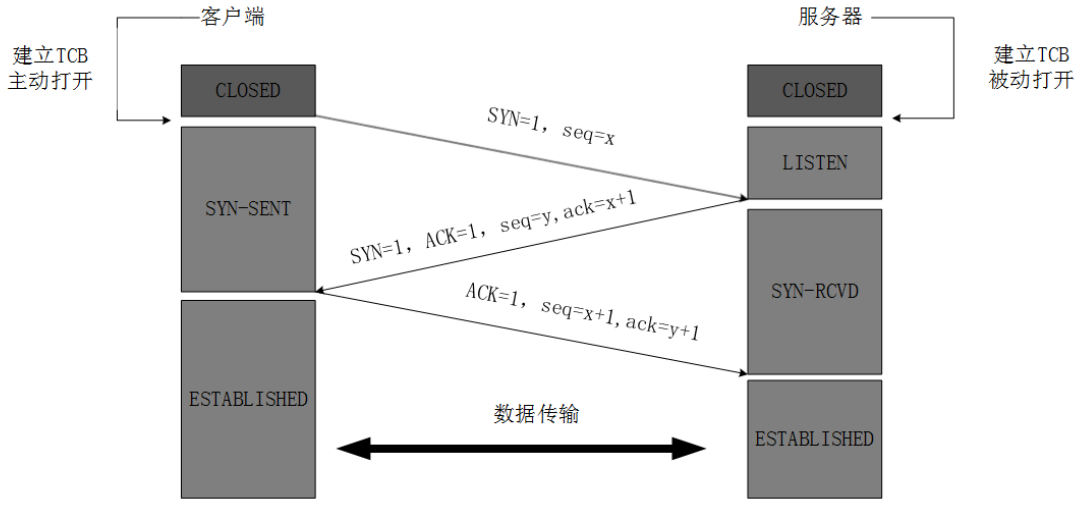

图 1: TCP 三次握手示意图

2. IPv6 隧道

隧道 (Tunnel) 技术是一种基于 IPv4 隧道来传送 IPv6 数据报文的封装技术。将 IPv6 包作为无结构意义的数据，封装在 IPv4 包中，如此穿越 IPv4 网络进行通信，并且在隧道 的两端可以分别对数据报文进行封装和解封装。隧道是一个虚拟的点对点的连接。隧道技术在定义上就是指包括数据封装、传输和解封装在内的全过程。下图展示了 IPv6 over IPv4 协议体系结构。隧道技术的实现需要有一个起点和一个终点。根据隧道终点的 IPv4 地址的获取方式不同可以将 IPv6 over IPv4 隧道分为手动隧道和自动隧道。

a) 手工配置隧道 (Confifigured Tunneling)

对每个 IPv6 分组，都事先手工配置它所对应的隧道的端点，主要是用于隧道封装所需的 IPv4 地址。

b) 自动配置隧道 (Automatic Tunneling)

分组中所包含的 IPv6 地址和/或路由的下一跳决定隧道的端点，主要是指用于隧道封装所需的 IPv4 地址。

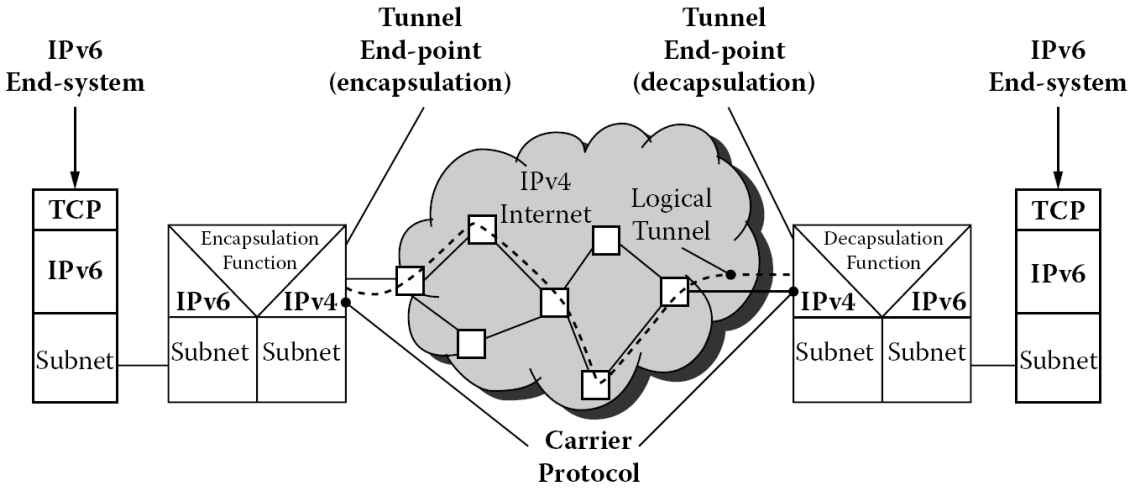

图 2: IPv6 over IPv4 or IPv6-in-IPv4 Tunneling

三、 实验内容

1. 基本操作

1）查看并记录 HostA 和 HostB 的 IPv6 (eth0 scope:link) 地址，命令 ifconfig eth0。

2） IPv6 连接：

HostA 的终端 1 中执行 sudo tcpdump -vxn host HostA 的 IPv6(eth0 scope:link) 地址 and HostB 的 IPv6(eth0 scope:link)地址 -i eth0。

HostA 的终端 2 中执行 ping6 -c 1 HostB 的 IPv6(eth0 scope:link)地址%eth0。

【实验要求】：此步骤需要记录终端 1 中抓到的 2 个报文数据，表明哪个是请求报文，哪个是回复报文。根据 IPv6 协议报文格式分析上述 tcpdump 抓包结果，要求画出 IPv6 以及 ICMPv6 的基本报头结构并将各个字段分别填入其中 (例如 IP 协议版本、源地址和目的地址、源端口和目的端口、报文含义等)。对这个报文的含义加以解释。

3）路由器公告报文

路由器公告报文的目的地址为 IPv6 组播地址中的“全节点地址”，它的地址是 FF02::1，可用 tcpdump 侦听路由器公告报文。HostA 中另开一个终端 3 用于侦听路由器公告报文，命令为：sudo tcpdump -vxn host ff02::1 -i eth0。

【实验要求】：记录 HostA 的终端 3 中 tcpdump 抓包得到的数据。由于路由器公告报文的发送有一定地周期，因此这里可能需要等待较长时间，可以把终端 3 最小化继续进行其他实验，等有结果后记录抓包得到的数据。如仍未顺利抓到此包，也可以参考附录中的图 4 回答下述问题。根据 tcpdump 抓取到的报文数据说明路由器通告报文通告了哪些信息并简单解释网络中的其它主机将会如何使用这些信息？

4）地址解析

地址解析的目的是通过对端的 IP 获取对端的 MAC 地址。由于地址解析过程会在数据发送前自动进行，因此需要先用 tcpdump 侦听，再 ping 对端，即可观察到 NS 和 NA 报文。

HostA 的终端 1 执行命令 sudo tcpdump -vxn host HostA 的 IPv6(eth0 scope:link) 地址 -i eth0。

HostA 的终端 2 执行命令 ping6 -c 1 HostB 的 IPv6(eth0 scope:link)地址%eth0。

【实验要求】：记录 HostA 的终端 1 中观察到邻居请求 (NS) 和邻居通告 (NA) 报文。根据抓取到的报文数据说明邻居请求及邻居公告报文通告了哪些信息，这些信息有什么作用？

5）TCP 三次握手

netcat 可以在主机间建立 TCP 连接，建立连接时，可以用 tcpdump 对报文抓包，观察到 TCP 的三次握手过程。

首先在 HostB 的终端 1 中执行 nc -l 1958 侦听 1958 端口。

之后在 HostA 的终端 1 执行 sudo tcpdump -vxn host HostA 的 IPv4 地址 and HostB 的 IPv4 地址 -i eth0。

在 HostA 的终端 2 执行 nc HostB 的 IPv4 地址 1958。

【实验要求】：完整记录 HostA 的终端 1 中观察到的前三个报文，即 TCP 握手报文。标注出每个报文的类型（SYN、SYN/ACK、ACK）。简要说明 TCP 协议采用三次握手的原因。说明实验中为何无法抓到 RST 包？请问编写应用程序时我们是否需要处理这些报文，为什么？

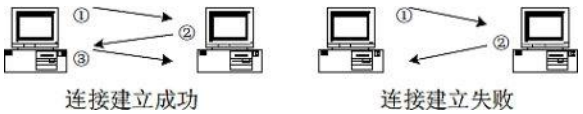

图 3: TCP 三次握手

2. 隧道

注意 隧道删除命令为 sudo ip tunnel del 隧道名称，若添加隧道命令打错可用此命令删除后重建。

1） 打开两个 Host 虚拟机并各打开一个终端

2） 在 HostA 的终端上执行命令：

sudo ip tunnel add sit1 mode sit remote HostB 的 IPv4 地址 local HostA 的 IPv4 地址 dev eth0（IPv4 地址可以由命令 ifconfig eth0 获得）

sudo ip link set sit1 up

ip link show up【记录结果，此时可以看到名字为 sit1 的设备】

sudo ip addr add 3ffe:3216:2101:2106:1234::A/80 dev sit1

ip tunnel show【记录结果】

3）在 HostB 的终端上执行命令：

sudo ip tunnel add sit1 mode sit remote HostA 的 IPv4 地址 local HostB 的 IPv4 地址 dev eth0（IPv4 地址可以由命令 ifconfig eth0 获得）

sudo ip link set sit1 up

ip link show up【记录结果，此时可以看到名字为 sit1 的设备】

sudo ip addr add 3ffe:3216:2101:2106:1234::B/80 dev sit1

ip tunnel show【记录结果】

4）在 HostA 上打开两个终端。其中一个终端用 tcpdump 侦听报文，另一个终端用于 ping6。

(a). 首先在 HostA 的第 1 个终端中执行命令 sudo tcpdump -vxn -i sit1。

(b). 其次在 HostA 的第 2 个终端中执行命令 ping6 -c 1 3ffe:3216:2101:2106:1234::B，检查是否可以 ping 通，若不通则需要检查之前的步骤是否正确完成。

(c). 记录 HostA 的第 1 个终端中由 tcpdump 抓取的前两个报文。

(d). 关闭 HostA 的这两个终端并重新打开两个新的终端。

(e). 在 HostA 的第 1 个终端中执行命令 sudo tcpdump -vxn host HostA 的 IPv4 地址 and HostB 的 IPv4 地址 -i eth0。

(f). 在 HostA 的第 2 个终端中执行命令 ping6 -c 1 3ffe:3216:2101:2106:1234::B。

(g). 记录 HostA 的第 1 个终端中由 tcpdump 抓取的前两个报文（可能要等一会儿）。

【实验要求】：记录需要记录的实验数据。从报文结构上看，通过隧道通信与两个 IPv6 主机直接通信的区别是什么，即上述隧道通信的报文有什么特点？

四、 实验报告

报告要求：电子版报告，内容包括实验题目、实验原理（对实验中用到的所有命令加以解释）、实验结果（实验中所有需要记录的结果，可截图，截图中需要包含命令、结果及虚拟机控制台界面上方的“用户名、实验名和虚拟机名称”）、问题回答、实验收获。

注意 实验报告及时提交到邮箱 ustc_infonet@163.com 中，第二次实验报告提交 PDF 文件，标题和文件命名规则为：姓名-学号-信网第二次实验，晚交会记录并扣分。

五、 附录

```
sa24006191@noble:~$ sudo tcpdump -vxn host ff02::1 -i ens33
[sudo] sa24006191 的密码：
tcpdump: listening on ens33, link-type EN10MB (Ethernet), snapshot length 262144 bytes
17:24:19.471157 IP6 (flowlabel 0x8307a, hlim 255, next-header ICMPv6 (58) payload length: 112
) fe80::eafc:afff:fefb:7fc3 > ff02::1: [icmp6 sum ok] ICMP6, router advertisement, length 112
        hop limit 64, Flags [none], pref medium, router lifetime 9000s, reachable time 0ms, r
etrans timer 0ms
          source link-address option (1), length 8 (1): e8:fc:af:fb:7f:c3
          mtu option (5), length 8 (1):  1500
          prefix info option (3), length 32 (4): 2001:da8:d800:c030::/64, Flags [onlink, auto
], valid time 172800s, pref. time 86400s
          rdnss option (25), length 40 (5):  lifetime 9000s, addr: fe80::eafc:afff:fefb:7fc3
addr: fe80::3e33:ff:fe51:fd40
          advertisement interval option (7), length 8 (1):  600000ms
        0x0000:  6008 307a 0070 3aff fe80 0000 0000 0000
        0x0010:  eafc afff fefb 7fc3 ff02 0000 0000 0000
        0x0020:  0000 0000 0000 0001 8600 1fb4 4000 2328
        0x0030:  0000 0000 0000 0000 0101 e8fc affb 7fc3
        0x0040:  0501 0000 0000 05dc 0304 40c0 0002 a300
        0x0050:  0001 5180 0000 0000 2001 0da8 d800 c030
        0x0060:  0000 0000 0000 0000 1905 0000 0000 2328
        0x0070:  fe80 0000 0000 0000 eafc afff fefb 7fc3
        0x0080:  fe80 0000 0000 0000 3e33 00ff fe51 fd40
        0x0090:  0701 0000 0009 27c0
```

图 4: 路由器公告报文参考图

实验三：IPv6 网络编程实验——入门

一、 实验目标

1. 了解 TCP 的 Client/Server 结构的连接建立过程。
2. 掌握网络 Socket 编程的基本概念和基本方法。
3. 掌握 TCP 的 Client/Server 结构的通信的编程。

二、 实验原理

1. TCP 通信

传输层和应用层之间进行的数据交换称为报文（Message），而在传输层和网络层之间进行交换的数据称为数据报（Datagram）。传输层可以使用传输控制协议（TCP）来封装数据。TCP 协议面向连接，使用字节流传送服务，是可靠的。

在面向连接的 Client/Server 结构中：服务器首先启动，通过调用 socket() 建立一个套接口，然后调用 bind() 将该套接口和本地网络地址联系在一起，再调用 listen() 使套接口做好侦听的准备，并规定它的请求队列的长度，之后就调用 accept() 来接收连接。客户在建立套接口后就可调用 connect() 和服务器建立连接。连接一旦建立，客户机和服务器之间就可以通过调用 read() 和 write() 来发送和接收数据。最后，待数据传送结束后，双方调用 close() 关闭套接口。

2. 套接字 socket

套接字（socket）是通信的基石，是支持 TCP/IP 协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：连接使用的协议，本地主机的 IP 地址，本地进程的协议端口，远地主机的 IP 地址，远地进程的协议端口。

应用层通过传输层进行数据通信时，TCP 会遇到同时为多个应用程序进程提供并发服务的问题。多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与 TCP / IP 协议交互提供了套接字 (Socket) 接口。应用层可以和传输层通过 Socket 接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。

Socket 连接，至少需要一对套接字，分为 clientSocket，serverSocket。连接分为 3 个步骤：

a) 服务器监听：服务器并不定位具体客户端的套接字，而是时刻处于监听状态。

b) 客户端请求：客户端的套接字要描述它要连接的服务器的套接字。提供地址和端口号，然后向服务器套接字提出连接请求。

c) 当服务器套接字收到客户端套接字发来的请求后，就响应客户端套接字的请求，并建立一个新的线程，把服务器端的套接字的描述发给客户端，一旦客户端确认了此描述，就正式建立连接。而服务器套接字继续处于监听状态，继续接收其他客户端套接字的连接请求。

三、 实验内容

1. 阅读本文档中的两个程序 TCPClient.c、TCPServer.c，见本文档附录 A。
2. 编译与运行程序：
   a) 新建文件 TCPClient.c、TCPServer.c，参考文档中的示例编写程序。
   虚拟机中创建文件的其中一种方式如下(也可以按照其它方式来做)：
   - 在要保存代码文件的文件夹中打开终端，输入 cat > TCPClient.c，然后输入代码内容，结束用 Ctrl+D；
   - 继续输入 cat > TCPServer.c，然后输入代码内容，结束用 Ctrl+D。

   b) 在保存代码文件的文件夹中打开两个终端。
   c) 首先在终端 1 编译运行服务器：
   I. 执行 gcc TCPServer.c -o TCPServer 编译服务器程序。
   II. 执行命令./TCPServer 运行服务器程序。
   d) 其次在终端 2 编译运行客户端：
   I. 执行 gcc TCPClient.c -o TCPClient 编译客户端程序。
   II. 执行命令./TCPClient ::1 运行客户端程序。
   注意 这里的::1 表示回环地址，即这个示例程序完成了同一个主机上两个进程之间的通信。
3. 请根据实验原理和[The GNU C Library Reference Manual]手册理解本实验示例代码的含义。

四、 思考题

1. 上述示例程序是一个基于 C/S 模型的 TCP 通信程序。请根据对示例程序的理解将 socket()、write()、close()、read()、listen()、bind()、connect()、accept() 这八个基本函数填入如图 1 所示的典型 C/S 模型 TCP 通信程序流程图中。

2. 图 2 给出了一张 TCP 状态转移图。请根据你的理解将执行上题中每个函数之后 TCP 连接所处的状态在图中标出，标识的方式是在状态旁写出 [主机 A/主机 B 执行函数名，主机 A/主机 B 处于此状态]。根据这个状态转移图的理解以及相关资料简要说明 SYN Flood 攻击的实现原理。

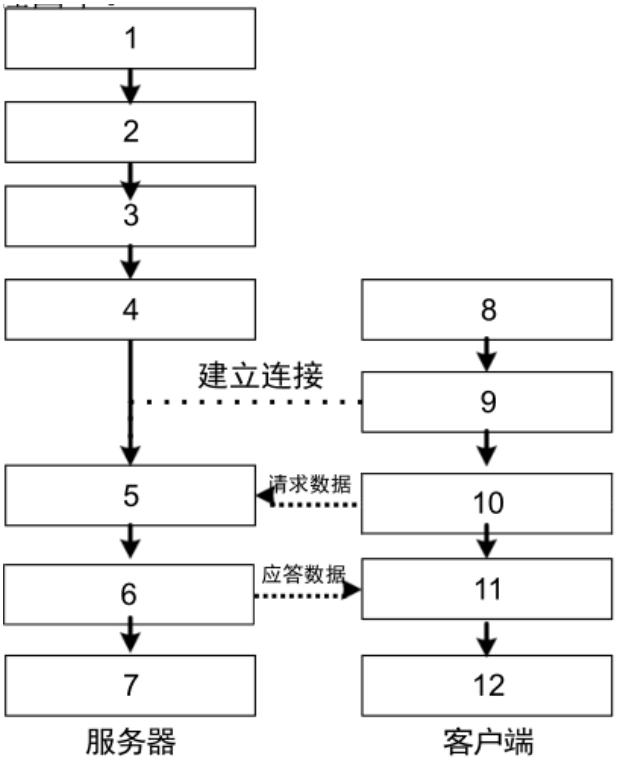

图 1: 基于 C/S 模型的 TCP 通信程序流程图

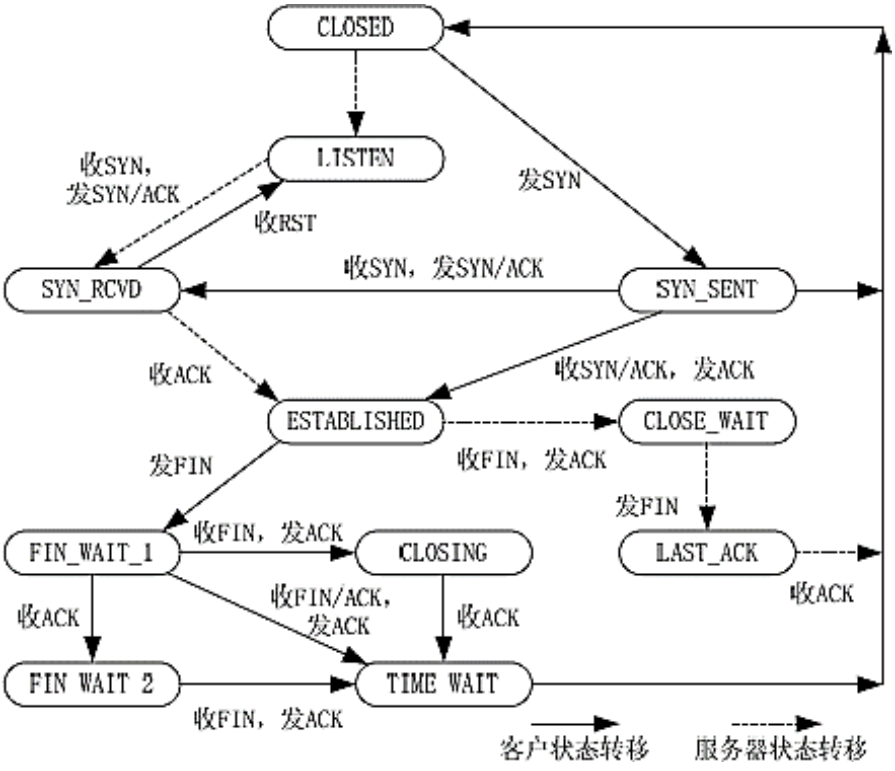

图 2: TCP 状态转移图

五、 实验报告

报告要求：电子版报告，内容包括实验题目、实验原理 (对示例代码中的语句进行解释)、实验结果(示例程序运行截图)、实验结果、实验收获。

注意 实验报告及时提交到邮箱 ustc_infonet@163.com 中，第三次实验报告提交 PDF 文件，标题和文件命名规则为：姓名-学号-信网第三次实验，晚交会记录并扣分。

实验四：IPv6 网络编程实验——自由设计

一、 实验目标

1. 了解多进程编程
2. 掌握基于 socket 的全双工通信
3. 掌握基于 socket 的文件传输

二、 实验原理

1. 多进程编程

进程 (process) 是操作系统的概念，为程序的一次执行在操作系统中或内存中的映像，同时伴随着资源的分配和释放。并具有以下特征的活动单元。

- 一组执行的指令序列
- 一个当前状态
- 相关的系统资源集合

Linux 内核通过唯一的进程标识符 PID 来标识每个进程。PID 存放进程描述符的 pid 字段中，新创建的 PID 通常是前一个进程的 PID 加 1，不过 PID 的值有上限。

使用 fork() 函数复制创建新进程：

```
1  //函数头文件及函数原型
2  #include <unistd.h>
3  pid_t fork(void);
```

在 Linux 中创建一个新进程的唯一方法是使用 fork() 函数。fork() 函数是 Linux 中一个非常重要的函数，用于从已存在的进程中创建一个新进程。新进程称为子进程，而原进程称为父进程。使用 fork() 函数得到的子进程是父进程的一个复制品，它从父进程处继承了整个进程的地址空间，包括进程的上下文、代码段、进程堆栈、内存信息、打开的文件描述符、符号控制设定、进程优先级、进程组号、当前工作目录、根目录、资源限制、信号处理方式和控制终端等，而子进程所独有的只有它的进程号、资源使用和计时器等。实际是在父进程中执行 fork() 函数时，父进程会复制一个子进程，而且父子进程的代码从 fork() 函数的返回开始分别在两个地址空间中同时运行，从而使两个进程分别获得所属 fork() 函数的返回值，其中在父进程中的返回值是子进程的进程号，而在子进程中返回 0，若出错返回 -1。

2. 全双工通信

双工（Full Duplex）是通讯传输的一个术语。通信允许数据在两个方向上同时传输，它在能力上相当于两个单工通信方式的结合。全双工指可以同时（瞬时）进行信号的双 5.3 实验内容向传输（AB 且 BA）。指 AB 的同时 BA，是瞬时同步的。

全双工方式在发送设备的发送方和接收设备的接收方之间采取点到点的连接，通信系统的每一端都设置了发送器和接收器，因此，能控制数据同时在两个方向上传送。全双工方式无需进行方向的切换，因此，没有切换操作所产生的时间延迟，这对那些不能有时间延误的交互式应用（例如即时聊天、远程监测和控制系统）十分有利。

三、 实验内容

本实验是在第 4 章实验三的基础之上的扩展实验。根据扩展功能的不同，本实验有两个题目，分别为：

- 基于 socket 的全双工通信编程
- 基于 socket 的文件传输编程

同学们可以在对实验三中基于 socket 的基础网络编程的充分理解之上，自行设计 Client/Server 结构通信中的扩展功能并实现。附录 A 中列出了可能会用到的功能的示例代码（C 语言版本）。

(一)题目 1

要求编写一个程序，实现两个主机之间的文本聊天通信。通信需要在 IPv6 环境下完成 (即这两个主机均使用 IPv6 地址)。

人数限制：此题目仅允许 1 个人单独完成。

基本要求：此程序不需要有图形界面，即图形界面不作为评分依据。两个主机之间的聊天由一方作为发起方开始，而后不断进行直到任意一方发送“END”结束。要求聊天过程中两人都可以随时向对方发送文本消息，即需要实现两个主机之间的全双工通信。如果提交的代码只能实现两个主机轮流发送文本消息给对方将会酌情扣分。

扩展要求：此题目扩展要求自拟，比如可以实现“群聊”、“@”、“文件传输”功能。选做扩展要求者根据所选择的扩展要求的实现难度以及完成质量酌情加分。

(二) 题目 2

要求编写一个程序，将一个不小于 20M 的文件（不一定是文本文件）从一个主机发送到另外一个主机，通信需要在 IPv6 环境下完成 (即这两个主机均使用 IPv6 地址)。

人数限制：此题目可以由 1-4 个成员共同完成。

基本要求：此程序不需要有图形界面，即图形界面不作为评分依据。主机 B 收到的由主机 A 发来的文件必须与主机 A 上的文件一致，即发送前后文件的校验值 (CRC32) 相同。需要可以控制文件传输的速度 (即在程序开始运行时询问用户以多大的速度发送文件，而后一直以给定的速率传输)。要求在文件发送开始前接收主机可以选择拒绝或者同意接收文件。

扩展要求：要求在文件发送过程中两个主机都能看到传输的进度，即已传大小、百分比、文件传输速度、预计传输时间等信息，在文件发送过程中接收主机可以选择暂停、继续、终止接收文件。完成此要求者根据所选择的扩展要求的实现难度以及完成质量酌情加分。

四、 实验报告

报告要求：电子版报告，内容包括实验题目、实验原理 (对相关功能的实现方式及程序代码进行必要的解释)、实验结果 (程序运行截图)、实验收获。实验代码及相应的可执行程序需要一并提交。如果采用脚本语言 (比如 Ruby、Python) 则无需提交可执行程序。

注意

- 以上两题任选一题完成即可。
- 可以采用你熟悉的任何编程语言实现上述要求。
- 提交的代码会经由代码 Review 工具检查，如果检查后发现两组或两个人的代码重合度高于 50% 则算作抄袭，抄袭与被抄袭者本实验记为 0 分。简单修改变量名称与语句顺序不能降低重合度！
- 实验报告、实验代码、可执行程序请打包成 ZIP 格式压缩包并于最后一次实验结束两周内提交。
- 实验报告及时提交到邮箱 ustc_infonet@163.com 中，压缩包和报告的标题和文件命名规则为：姓名-学号-信网第四次实验，晚交会记录并扣分。

附录 A　实验 3.1 参考代码

TCPServer.c

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define MAXLINE 1024
#define TRUE    1

int main(int argc, char **argv)
{
    int sockfd, fd, n, m;
    char line[MAXLINE + 1];
    struct sockaddr_in6 servaddr, cliaddr;
    time_t t0 = time(NULL);
    printf("time #: %ld\n", t0);
    fputs(ctime(&t0), stdout);

    if((sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0)
        perror("socket error");

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin6_family = AF_INET6;
    servaddr.sin6_port = htons(20000);
    servaddr.sin6_addr = in6addr_any;

    if(bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) == -1)
        perror("bind error");

    if(listen(sockfd, 5) == -1)
        perror("listen error");

    while(TRUE) {

        printf("> Waiting clients ...\r\n");

        socklen_t clilen = sizeof(struct sockaddr);
        fd = accept(sockfd, (struct sockaddr*)&cliaddr, &clilen);
        if(fd == -1)
        { perror("accept error");
        }
```

```
      printf("> Accepted.\r\n");

while((n = read(fd, line, MAXLINE)) > 0)
         { line[n] = 0;
         if(fputs(line, stdout) == EOF)
            perror("fputs error");
      }
      close(fd);
   }

   if(n < 0) perror("read error");
   exit(0);
   }
```

TCPClient.c

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define MAXLINE 1024
#define TRUE    1

int main(int argc, char **argv)
{
    int sockfd, n, m;
    char line[MAXLINE + 1];
    struct sockaddr_in6 servaddr;
    time_t t0 = time(NULL);
    printf("time #: %ld\n", t0);
    fputs(ctime(&t0), stdout);

    if(argc != 2)
        perror("usage: a.out <IPaddress>");

    if((sockfd = socket(AF_INET6, SOCK_STREAM, 0)) < 0)
        perror("socket error");

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin6_family = AF_INET6;
    servaddr.sin6_port = htons(20000);

    if(inet_pton (AF_INET6, argv[1], &servaddr.sin6_addr) <= 0)
        perror("inet_pton error");

    if(connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
```

```
        perror("connect error");

    while(fgets(line, MAXLINE, stdin) != NULL)
        { send(sockfd, line, strlen(line), 0);
    }

    close(sockfd);

    exit(0);
}
```

附录B　网络编程参考资料

说明：以下内容均节选自 [The GNU C Library Reference Manual, Sandra Loosemore et.al]，此手册已放于 Host 虚拟机桌面。

附录 B1. 常用网络编程函数及数据结构

名称	功能	页码
[Function] socket	This function creates a socket and specifies communication style ***style***.	489
[DataType] sockaddr_in6	This is the data type used to represent socket addresses inthe Internet namespace.	474
[Function] listen	The listen function enables the socket ***socket*** to accept connections, thus making it a server socket.	493
[Function] bind	The bind function assigns an address to the socket ***socket***.	469
[Function] accept	This function is used to accept a connection request on the server socket ***socket***.	494
[Function] connect	The connect function initiates a connection from the socket with file descriptor ***socket*** to the socket whose address is specified by the ***addr*** and ***length*** arguments.	492
[Function] shutdown	The shutdown function shuts down the connection of socket ***socket***.	490

附录 B2. 网络字节顺序及转换函数

Different kinds of computers use different conventions for the ordering of bytes within a word. Some computers put the most significant byte within a word first (this is called "big-endian" order), and others put it last ("little-endian" order). So that machines with different byte order conventions can communicate, the Internet protocols specify a canonical byte order convention for data transmitted over the network. This is known as network byte order. When establishing an Internet socket connection, you must make sure that the data in the **sin_port** and **sin_addr** members of the **sockaddr_in** structure are represented in network byte order. If you are encoding integer data in the messages sent through the socket, you should convert this to network byte order too. If you don't do this, your program may fail when running on or talking to other kinds of machines. [pp.485]

名称	功能	页码
[Function] htons	This function converts the uint16_t integer ***hostshort*** from host byte order to network byte order.	486
[Function] ntohs	This function converts the uint16_t integer ***netshort*** from network byte order to host byte order.	486
[Function] htonl	This function converts the uint32_t integer ***hostlong*** from host byte order to network byte order. This is used for IPv4 Internet addresses.	486
[Function] ntohl	This function converts the uint32_t integer ***netlong*** from network byte order to host byte order. This is used for IPv4 Internet addresses.	486

附录 B3. 常用 IP 地址转换函数

名称	功能	页码
[Function] inet_addr	This function converts the IPv4 Internet host address name from the standard numbers-and-dots notation into binary data.	478
[Function] inet_aton	This function converts the IPv4 Internet host address name from the standard numbers-and-dots notation into binary data and stores it in the struct in_addr that ***addr*** points to.	477
[Function] inet_ntoa	This function converts the IPv4 Internet host address addr to a string in the standard numbers-and-dots notation.	478
[Function] inet_pton	This function converts an Internet address (either IPv4 or IPv6) from presentation (textual) to network (binary) format.	479
[Function] inet_ntop	This function converts an Internet address (either IPv4 or IPv6) from network (binary) to presentation (textual) form.	479

附录 B4. 字节处理函数

名称	功能	页码
[Function] memcpy	The memcpy function copies ***size*** bytes from the object beginning at ***from*** into the object beginning at ***to***.	118
[Function] memset	This function copies the value of ***c*** (converted to an unsigned char) into each of the first ***size*** bytes of the object beginning at ***block***.	121
[Function] memcmp	The function memcmp compares the ***size*** bytes of memory beginning at ***a1*** against the ***size*** bytes of memory beginning at ***a2***.	130
[Function] memmove	memmove copies the ***size*** bytes at ***from*** into the ***size*** bytes at ***to***, even if those two blocks of space overlap.	120
[Function] bzero	This is a partially obsolete alternative for memset, derived from BSD. Note that it is not as general as memset, because the only value it can store is zero.	123
[Function] bcopy	This is a partially obsolete alternative for memmove, derived from BSD. Note that it is not quite equivalent to memmove, because the arguments are not in the same order and there is no return value.	123

附录 B5. 文件描述符操作函数

名称	功能	页码
[Function] open	The open function creates and returns a new file descriptor for the file named by ***filename***.	356
[Function] close	The function close closes the file descriptor ***filedes***.	358
[Function] write	The write function writes up to ***size*** bytes from ***buffer*** to the file with descriptor ***filedes***.	361
[Function] send	The send function is like write, but with the additional flags ***flags***.	495
[Function] recv	The recv function is like read, but with the additional flags ***flags***.	496